VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELGAVI, KARNATAKA -590 018

A Mini-Project Report on

"3D HELICOPTER GAME"

Submitted in partial fulfillment for the Computer Graphics Laboratory with Mini-Project (18CSL67) course of Sixth Semester of Bachelor of Engineering in Computer Science & Engineering during the academic year 2020-21.

By

Team Code: **CGP2021-C14**

SREYAS P	**4MH18CS107**
SATHWIK N	**4MH18CS098**

: Under the Guidance of:

Prof. PRATAP M S

Assistant Professor

Department of CS&E

MIT Mysore

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MAHARAJA INSTITUTE OF TECHNOLOGY MYSORE

Belawadi, S.R. Patna Taluk, Mandya Dist-571477.

Accredited By:

2020-21

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MAHARAJA INSTITUTE OF TECHNOLOGY MYSORE

~~ CERTIFICATE ~~

*Certified that the mini-project work entitled "**3D HELICOPTER GAME**" is a bonafide work carried out by **Sreyas P (4MH18CS107)** & **Sathwik N (4MH18CS098)** for the Computer Graphics Laboratory with Mini-Project (18CSL67) of Sixth Semester in Computer Science & Engineering under Visvesvaraya Technological University, Belgavi during academic year 2020-21.*

It is certified that all corrections/suggestions indicated for Internal Assignment have been incorporated in the report. The report has been approved as it satisfies the course requirements.

_____	_____
Signature of Guide	Signature of the HOD
Prof. Pratap M S	**Dr. Shivmurthy R C**
Assistant Professor	Professor & HOD
Dept. of CS&E	Dept. of CS&E
MIT Mysore	MIT Mysore

External viva

Name of the Examiners	**Signature with date**

1)……

2) ……

~~~~ ACKNOWLEDGEMENT ~~~~

It is the time to acknowledge all those who have extended their guidance, inspiration, and their wholehearted co-operation all along our project work.

We are also grateful to **Dr. B G Naresh Kumar**, principal, MIT Mysore and **Dr. Shivmurthy R C**, HOD, CS&E, MIT Mysore for having provided us academic environment which nurtured our practical skills contributing to the success of our project.

We wish to place a deep sense of gratitude to all Teaching and Non-Teaching staffs of Computer Science and Engineering Department for whole-hearted guidance and constant support without which Endeavour would not have been possible.

Our gratitude will not be complete without thanking our parents and our friends, who have been a constant source of support and aspirations

Sreyas P

Sathwik N

~~~ ABSTRACT ~~~

Our project titled “3D Helicopter Game” is an effort to combine some of the concepts and tools that we have learnt from the course on Computer Graphics with some of our own ideas. It is our rendition of a popular game with some fundamental changes. It involves navigating a 3D helicopter through a series of hoops or rings. Points are awarded for every successful passage through a ring. It has been implemented using OpenGL with the C++ programming language. There is much scope for improvement with respect to the elements of the game and the way it is played. Some suggested changes include making the background better, a different model for the helicopter, or adding a new functionality like shooting from the helicopter.

~~~~~ CONTENTS ~~~~~

CHAPTER - 1

INTRODUCTION

1.1 Aim

The main aim of this Mini Project is to illustrate the 3 D Helicopter game in OpenGL. It is a button game where Helicopter is navigated through a ring and points counted every time it successfully passes through it. Speed of the Helicopter increases with respect to the level we have reached.

1.2 Overview

Our project consists of an 3D helicopter and n number of rings. It involves navigating a 3D helicopter through a series of hoops or rings. Points are awarded for every successful passage through a ring. Helicopter can be navigated to up, down, right, left direction using keyboard functions. It has been implemented using OpenGL with the C++ programming language.

1.3 Outcome

Using the OpenGL APIs, a simple game is developed with colorful background of sky and grass with 3D helicopter that can be navigated through keyboard events in the direction of up, down, left and right where by default helicopter moves forward. On every successful entry into the ring points are awarded and levels are increased.

CHAPTER – 2

DESIGN AND IMPLEMENTATION

2.1 Algorithm

Step1: Start

Step2: Initially we have to declare some Variables

Step3: Now we will set a view volume to a perspective one by using gluPerspective API

i.e, **gluPerspective (theta, aspect, Dnear, Dfar);**

Step4: To Draw a helicopter in the Scene, define set functions they are

- drawOval ( )
- drawBlade ( )
- drawmyRotor ( )
- helibody ( )
- drawHelicopter ( )

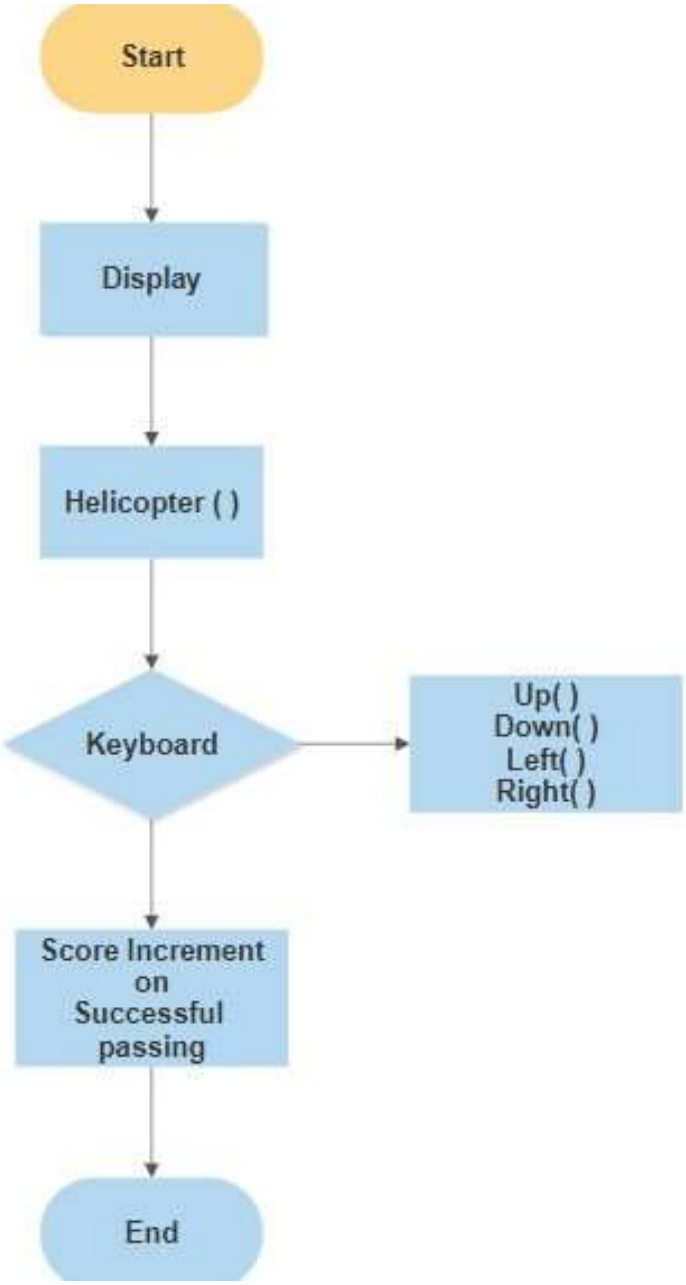
Step5: Now we Should draw the Scene Such that the helicopter should pass through n number of rings, for that we will define a function drawScene( )

Step6: We use a ReshapeFunc which has a callBack function which is called whenever size or shape of the application window changes. It takes 2 arguments; they are width and height of reshaped window. i.e we are setting a viewport

Step7: User will control the helicopter by navigating left, right, Top and bottom through a KeyBoardFunc

Step8: Stop

2.2 Flow Chart



2.3 OpenGL API's Used with Description

- **glutInit( ):** glutInit will initialize the GLUT library and negotiation a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.
- **glutInitDisplayMode( ):** The initial display mode is used when creating toplevel windows, sub windows, and overlays to determine the OpenGl display mode for the created window or overlay.

  GLUT_RGB specifies color (Red,Green,Blue)

  GLUT_DEPTH allocates Z-buffer for hidden Surface Elimination

  GLUT_DOUBLE uses two frame buffers one is used for drawing and another for the display
- **glutInitWindowSize(int width,int height):** It sets the intial width and height of the drawing region.
- **glutInitWindowPosition(int x,int y):** It sets initial position of Window on the Monitor.we can customize the position by giving X and Y values.
- **glutCreateWindow(**): It creates awindow with name and it returns an identifier for the window
- **glutDisplayFunc():** It sets the function that draws to the screen. This display function should basically handle all the GL drawing commands.hence all the glut Programs set up a display function.
- **glutKeyboardFunc (void (*func)(unsigned char key, int x, int y)):** It sets the function that handles keystrokes occurring while the GL window is active. The inputs are the key pressed and. For example, if (key=='Q') ... or if (key=='f') ..., for instance.
- **glutReshapeFunc (void (*func)(int width, int height)):** It sets the function that handles user changes to the GL window size. The inputs are the new user-set window width and height.
- **glutManiLoop ( ):** It displays the GLUT windows and starts the user interface loop(which includes drawing to the screen and handling events through the idle function,keyboard function etc.
- **glutPostRedisplay ( ):** It notes that the GL window needs to be redrawn. The display function should not be called directly, because multiple glutPostRedisplay() calls can

be combined into one, helping eliminate redundant redraws (which would slow interactivity).

- **glClear( ):** It takes a single argument(mask) that is the bitwise OR of several values indicating which buffer is to be cleared.
- **glMatrixMode():** It sets the current matrix mode i.e it specifies which matrix stack is the target for subsequent matrix operations.There are three modes they are

  GL_MODELVIEW: Applies subsequent matrix operations to the modelview matrix stack.

  GL_PROJECTION: Applies subsequent matrix operations to the projection matrix stack.

  GL_TEXTURE: Applies subsequent matrix operations to the texture matrix stack.
- **glLoadIdentity( ):** This function replaces the current matrix with identity matrix.It is called whenever the glMatrixmode( ) is called.
- **glScalef(Glfloat x, Glfloat y, Glfloat z):** It is used for resizing the object according to the requirement of the scene.This function provides a general scaling along the x, y , and z axes. The 3 arguemnts indicate the desired scale factors along each of the three axes.
- **gluLookAt(X0, Y0, Z0, Xref, Yref, Zref, Vx, Vy, Vz):** It is used to set the position of the camera, orientation, and updirection and the reference point where the camera has to orient to.By Default, gluLookAt(0, 0, 0, 0, 0, -1, 0, 1, 0).
- **glutSwapBuffers( ):** It is the last function called in the windows display callback, performs the actual buffer swap, Specifically it promotes the contents of back buffer of the layer in use of current window to become the contents of the front buffer.
- **glPushMatrix( ):** It pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.
- **glPopMatrix( ):** It pops the current matrix stack, replacing the current matrix with the one below it on the stack.

- **glTranslatef( ):** It produces a translation by x y z . The translation vector is used to compute 4*4 translation matrix and the current matrix is multiplied by this translation matrix, with the product replacing the current matrix.
- **glRotatef( ):** It produces a rotation of angle degrees around the vector x y z. The current matrix (see glMatrixMode) is multiplied by a rotation matrix with the product replacing the current matrix.
- **glutSolidSphere (GL_double radius, GL_int slices, GL_int stacks):** It draws a shaded sphere centered at the origin and it takes 3 parameters

  Radius: The radius of the sphere

  Slices: The number of subdivisions around the Z-axis (similar to lines of longitude)

  Stacks: The number of subdivisions along the Z-axis (similar to lines of latitude)
- **glutSolidTorus(GL_double innerRadius, GL_double outerRadius, GL_int sides, GL_int rings):** It renders a solid or wireframe torus(doughnut) respectively centered at modelling coordinates origin whose axis is aligned with the Z-axis.

  innerRadius: It is the inner radius of torus.

  outerRadius: It is the outer radius of torus.

  Sides: Number of sides for each radial section.

  Rings: Number of radial divisions for the torus.
- **glRasterPos3f(GLfloat x, GLfloat y, GLfloat z):** It is used to position pixel and bitmap write operations. It is maintained with subpixel accuracy, the current raster position consists of three window coordinates(x, y, z).
- **gluPerspective(theta, aspect, Dnear, Dfar):** It specifies a viewing volume into the world coordinate system. It takes 4 parameters theta value,aspect ratio, Dnear value and Dfar value.In general aspect ratio in gluPerspective should match the aspect ratio of the associated viewport
- **gluNewQuadric( ):** Creates a new quadric object and returns a pointer to it.
- **gluDeleteQuadric( ):** It destroys the quadrics object (created with **gluNewQuadric()**) and frees any memory it uses.

- **gluCylinder(qobj, baseRadius, topRadius, height, slices, stacks):** This function draws a cylinder oriented along the z-axis.The base of the radius is placed at z=0, and the top at z=height.The parameters are

  qobj: the Quadric object (created with gluNewQuadric( )).

  baseRadius: The radius of the cylinder at z=0.

  topRadius: The radius of the cylinder at z=height.

  Height: height of the cylinder

  Slices: Number of subdivisions around the z-axis.

  Stacks: Number of subdivisins along the z-axis.

- **gluQuadricDrawStyle(qObj, GLU_FILL):** It specifies the draw style for quadrics rendered with quad .

  qObj : the Quadric object (created with gluNewQuadric( )).

  GLU_FILL: Quadrics are rendered with polygon primitives. The polygons are drawn in a counterclockwise fashion with respect to their normals (as defined with gluQuadricOrientation).

- **glViewport(GLint x, GLint y, GLsizei width,GLsizei height):** Set the viewport to current window size.

  x, y: Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0).

  Width and height: Specify the width and height of the viewport. When a GL context is first attached to a window, width and height are set to the dimensions of that window.

- **glEnable(GL_DEPTH_TEST):** When depth testing is enabled, OpenGL tests the depth value of a fragment against the content of the depth buffer. OpenGL performs a depth test and if this test passes, the fragment is rendered and the depth buffer is updated with the new depth value. If the depth test fails, the fragment is discarded.

2.4 Source Code

```
#include <windows.h>
#include <GL/glut.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

const GLfloat ViewAngleOffset=0.01;
const GLfloat ZoomRadiusOffset=0.05;

const GLfloat SIZE1= 1000;
const GLfloat NEAR_Z= 0.001;
const GLfloat FAR_Z= 100.0;
const GLfloat PI= 3.14159265;
void drawOval(GLfloat x, GLfloat y, GLfloat z);
void drawSphere(float fRadius, GLint slices, GLint rings);
void drawCylinder(float fTopR, float fBottomR, float fHeight);
GLfloat deg2rad(GLfloat degree);
void drawHelicopter();
void drawScene();
void init (void);
void showMenu(void);
void keyboardCallbackProc(unsigned char key, int x, int y);
const GLfloat ViewStartRadius=5;
const GLfloat ViewStartTheta=60;
const GLfloat ViewStartPhi=90;
int touchx=0;
int lev=1;
int m=0;
int rings[][3]={{-50,0,0},{-100,4,4},{-150,2,-4},{-200,6,6},{-250,4,-6}};
```

```
int kount=0;
int start=1;
const GLfloat MovementSpeed=0.3;

GLfloat mainRotorSpinSpeed=0;
GLfloat mainRotorRotatingAngle=0;
bool bCamera=false;
GLfloat viewPortw, viewPorth;

bool bWire;
GLfloat fovy = 90.0;
int H= SIZE1, W= SIZE1;

enum {PERSPECTIVE} ePrjn = PERSPECTIVE;
GLfloat move_x=0, move_z=0,move_y=0;
//the polar coordinate parameter
GLfloat theta=deg2rad(ViewStartTheta), phi=deg2rad(ViewStartPhi);

GLfloat radius=7;
//ViewStartRadius;
GLfloat deg2rad(GLfloat degree)
{
        return degree/360.0*2*PI;
}
void showMenu(void)
{
        printf("\n\n\n\n3D HELICOPTER GAME");
        printf("\n-------------------------------------------------------------");

        printf("\n Use m/M to see the menu again.");
```

```
        printf("\n Use escape key to exit.\n");
    printf("\nSCORE=%d",touchx);
    printf("\nMOVE_x=%f",move_x);
    printf("\nMOVE_y=%f",move_y);
    printf("\nMOVE_z=%f",move_z);
    printf("\n m=%d",m);
    printf("\nLEVEL=%d",lev);

}
void init (void)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if(ePrjn == PERSPECTIVE)
        {
                gluPerspective(fovy, (GLfloat)W/(GLfloat)H, NEAR_Z, FAR_Z);
        }
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        glClearColor (0.0, 0.8, 1, 0.3);
        glShadeModel(GL_SMOOTH);
        glEnable(GL_DEPTH_TEST);

        bWire = false;
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        showMenu();
}

void keyboardCallbackProc(unsigned char key, int x, int y)
        // This is the callback procedure for capturing OpenGL Keyboard events.
```

```
{
        GLfloat temp;
        switch(key)
        {
        case ' ':
                bCamera=true;
                break;
        case 'L':
        case 'l':
                move_z-=MovementSpeed;
                break;
        case 'J':
         case 'j':
                move_z+=MovementSpeed;
                break;
        case 'K':
         case 'k':
                  move_y-=MovementSpeed;
                  break;

    case 'I':
    case 'i':
      move_y+=MovementSpeed;

      break;
        case 27 :
                exit(1);
                break;
        }
        glutPostRedisplay();
```

```
}
void reShapeCallbackProc(int w, int h)
{
        glViewport(0, 0, w, h);
        W = w;
        H = h;
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if(ePrjn == PERSPECTIVE)
        {
                gluPerspective(fovy, (GLfloat)W/(GLfloat)H, NEAR_Z, FAR_Z);
        }
        glMatrixMode(GL_MODELVIEW);
        glutPostRedisplay();
}
void drawCylinder(float fTopR, float fBottomR, float fHeight)
{
        GLUquadricObj* pObj;
        pObj = gluNewQuadric();

        gluQuadricDrawStyle(pObj, GLU_FILL);
        gluCylinder(pObj, fTopR, fBottomR, fHeight, 20, 20);
                gluDeleteQuadric(pObj);
}
void renderBitmapString(float x,float y,float z,void *font,char*string)
{
char *c;
glRasterPos3f(x, y,z);
for(c=string; *c != '\0'; c++)
{
glutBitmapCharacter(font, *c);
```

```
}
}
void drawScene()
{
        int i,k;
        const GLfloat GroundHeight=-10;
        const GLfloat View_Size=350;
        glPushMatrix();
                glColor4f(0.0, 0.8, 0.0,1);
                glLineWidth(2);
                        glBegin(GL_POLYGON);
                        glVertex3f(-View_Size, GroundHeight,-View_Size);
                        glVertex3f(View_Size, GroundHeight,-View_Size);
                        glVertex3f(View_Size, GroundHeight, View_Size);
                        glVertex3f(-View_Size, GroundHeight, View_Size);
                        glEnd();

        glColor4f(1,0,0.7,1);
                glPushMatrix();
                        glTranslatef(-20, 2, -6);
                                glPushMatrix();
            glTranslated(0,0,7);
            renderBitmapString(0, -0.5/1.3,0,GLUT_BITMAP_HELVETICA_18,"LEVEL  ");
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, (lev+48));
          glPopMatrix();
                glPopMatrix();
        glPopMatrix();

for(k=0;k<5;k++)
{
```

```
glPushMatrix();
glTranslated(rings[k][0],rings[k][1],rings[k][2]);
        glColor4f(lev*0.2,0.9-k*0.2,0.1+k*0.2,1);
        glRotatef(90,0,1,0);
   glScaled(3,3,3);
        glutSolidTorus(0.2,2, 30, 30);
glPopMatrix();
}
}
void drawOval(GLfloat x, GLfloat y, GLfloat z)
{
        glPushMatrix();
        glScalef(x, y, z);
        glutSolidSphere(1.0, 20, 20);

        glPopMatrix();
}
void drawBlade()
{       const GLfloat BladeWidth=0.1;
        const GLfloat BladeHeight=0.08;
        const GLfloat BladeLength=1.0;
        glPushMatrix();
                glTranslatef(0, 0, BladeLength);
      drawOval(BladeWidth, BladeHeight, BladeLength);
        glPopMatrix();
}
void drawMyRotor()
{
 glPushMatrix();
 glPushMatrix();
```

```
glColor4f(1,1,1,0.11);
 glTranslated(0,.4,0);
 drawBlade();
 glRotated(120,0,1,0);
 drawBlade();
 glRotated(120,0,1,0);
 drawBlade();
 glPopMatrix();
glTranslated(0,.4,0);
 glRotated(90,1,0,0);
glColor4f(0,0.5,1,1);
 drawCylinder(.08,.08,.20);
glPopMatrix();
}
void helibody()
{
 glPushMatrix();
 glTranslated(1.4,0,0);
 glColor4f(1,.7,0,.1);
 drawOval(.08,.08,.08);
 glPopMatrix();
 glPushMatrix();
 glTranslated(.5,0,0);
 glRotated(90,0,1,0);
 glScaled(.6,.8,.8);
 glColor4f(0,0.5,1,1);
 drawCylinder(.2,.1,1.0);
 glPopMatrix();
 glPushMatrix();
 glColor4f(1,.7,0,1);
```

```
 drawOval(0.7,0.4,0.4);
 glPopMatrix();
 glPushMatrix();
 glTranslated(-.31,.068,0);
 glRotated(-5,0,0,1);
 glColor4f(0,1,1,.3);
 drawOval(0.33,0.3,0.3);
 glPopMatrix();
 glPushMatrix();
 glTranslated(0,.3,0);
 glScaled(.5,.5,.5);
 glRotatef(mainRotorRotatingAngle, 0,1,0);
 drawMyRotor();
 glPopMatrix();
}
void drawHelicopter()
{
        glPushMatrix();
   glScaled(2,2,2);
        helibody();
        glPopMatrix();
}
void displayCallbackProc (void)
{
        GLfloat eye_x, eye_y, eye_z;
        const GLfloat X_Offset=1.5;
        const GLfloat Y_Offset=-1;
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(1.0, 1.0, 1.0);
   mainRotorSpinSpeed=15;
```

```
        mainRotorRotatingAngle+=mainRotorSpinSpeed;
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        eye_x=radius*sin(theta)*sin(phi);
        eye_y=radius*cos(theta);
        eye_z=radius*sin(theta)*cos(phi);
        if (bCamera)
        {
                gluLookAt(eye_x+move_x, eye_y, eye_z+move_z, move_x, 0, move_z, 0, 1,
0);
 renderBitmapString(move_x,0+4,move_z-4,GLUT_BITMAP_HELVETICA_18,"SCORE : ");
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, ((touchx)+48));
 renderBitmapString(move_x,0+3.8,move_z-4,GLUT_BITMAP_HELVETICA_18," LEVEL :
");
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, ((lev/10)+48));
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, ((lev%10)+48));

                glTranslatef(-eye_x, -eye_y, -eye_z);
                drawScene();
                //translate helicopter a little back so that pilot is at rotation axis
                glTranslatef(X_Offset+move_x, Y_Offset+move_y,move_z);
   move_x-=MovementSpeed*0.1*lev;
     if(move_x<=(1.5+rings[m][0])&&move_x>=(rings[m][0]+1.30))
     if(move_y<=(3.60+rings[m][1])&&move_y>=(rings[m][1]-3.60))
     if(move_z<=(3.80+rings[m][2])&&move_z>=(rings[m][2]-3.80))
     if(++kount==1)
     touchx++;
     if(move_x<=rings[m][0]&&move_x>=(rings[m][0]-0.2))
     {m++;
     kount=0;
     }
```

```
        if(move_x<=-280)
        {

        if(touchx>=3)
        {
        lev++;
        }
        move_x=30;
        m=0;
        touchx=0;
        }
        glScalef(2, 2, 2);
                    drawHelicopter();
            }
            else{
            gluLookAt(eye_x+move_x, eye_y, eye_z+move_z+8, move_x, 0, move_z, 0, 1, 0);

    glScaled(1,1,1);

renderBitmapString(move_x,5,move_z+3,GLUT_BITMAP_HELVETICA_18,"HELICOPTER
GAME");
  renderBitmapString(move_x,3,move_z+4,GLUT_BITMAP_HELVETICA_18,"DEVELOPED
BY:");

  renderBitmapString(move_x,2,move_z,GLUT_BITMAP_HELVETICA_18,"SREYAS
P(4MH18CS107)");
  renderBitmapString(move_x,1,move_z,GLUT_BITMAP_HELVETICA_18,"SATHWIK
N(4MH18CS098)");

renderBitmapString(move_x,-1,move_z+3,GLUT_BITMAP_HELVETICA_18,"SPACE TO
CONTINUE");
```

```
renderBitmapString(move_x,-2,move_z+3,GLUT_BITMAP_HELVETICA_18,"PRESS I FOR
UP");

renderBitmapString(move_x,-3,move_z+3,GLUT_BITMAP_HELVETICA_18,"PRESS K FOR
DOWN");

renderBitmapString(move_x,-4,move_z+3,GLUT_BITMAP_HELVETICA_18,"PRESS J FOR
LEFT");

renderBitmapString(move_x,-5,move_z+3,GLUT_BITMAP_HELVETICA_18,"PRESS L  FOR
RIGHT");

renderBitmapString(move_x,-6,move_z+3,GLUT_BITMAP_HELVETICA_18,"PRESS ESC
TO EXIT");

        }

        glutSwapBuffers();

        glutPostRedisplay();

}

int main (int argc, char *argv[])

{       glutInit(&argc, argv);

        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

        glutInitWindowSize(SIZE1, SIZE1);

        glutInitWindowPosition(10, 10);

        glutCreateWindow("Helicopter game");

        init();

        glutDisplayFunc(displayCallbackProc);

        glutKeyboardFunc(keyboardCallbackProc);

        glutReshapeFunc(reShapeCallbackProc);

        glutMainLoop();

        return 1;

}
```

CHAPTER – 3

RESULT ANALYSIS

3.1 Snap Shots

Fig 3.1.1: Navigations

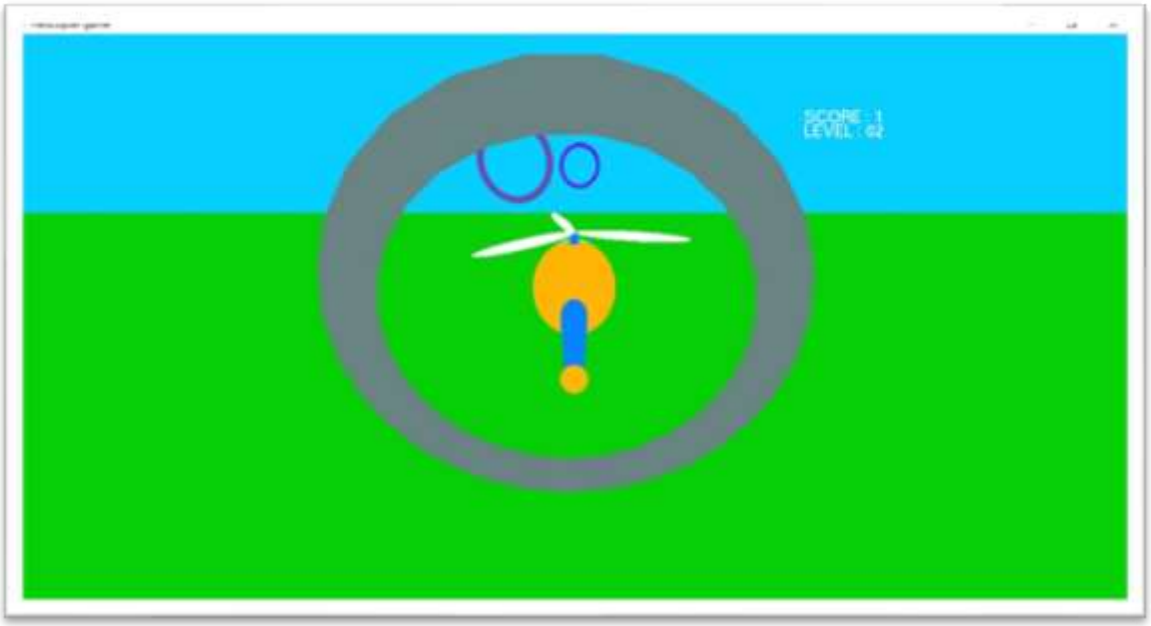


Fig 3.1.2: Helicopter and Rings

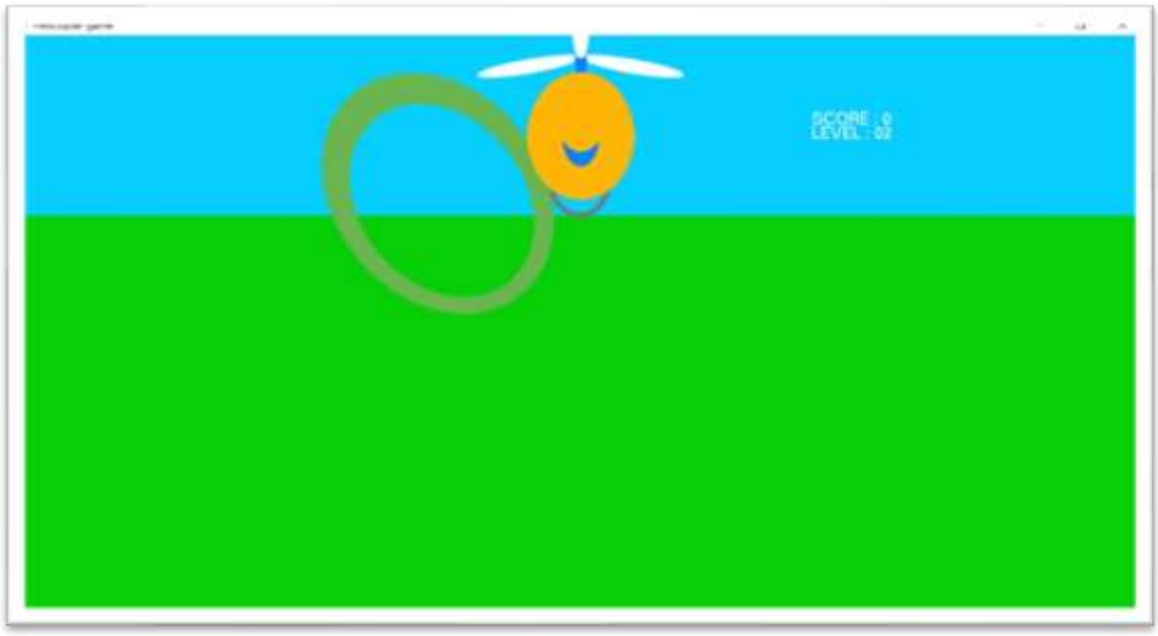


Fig 3.1.3: Upside navigated.

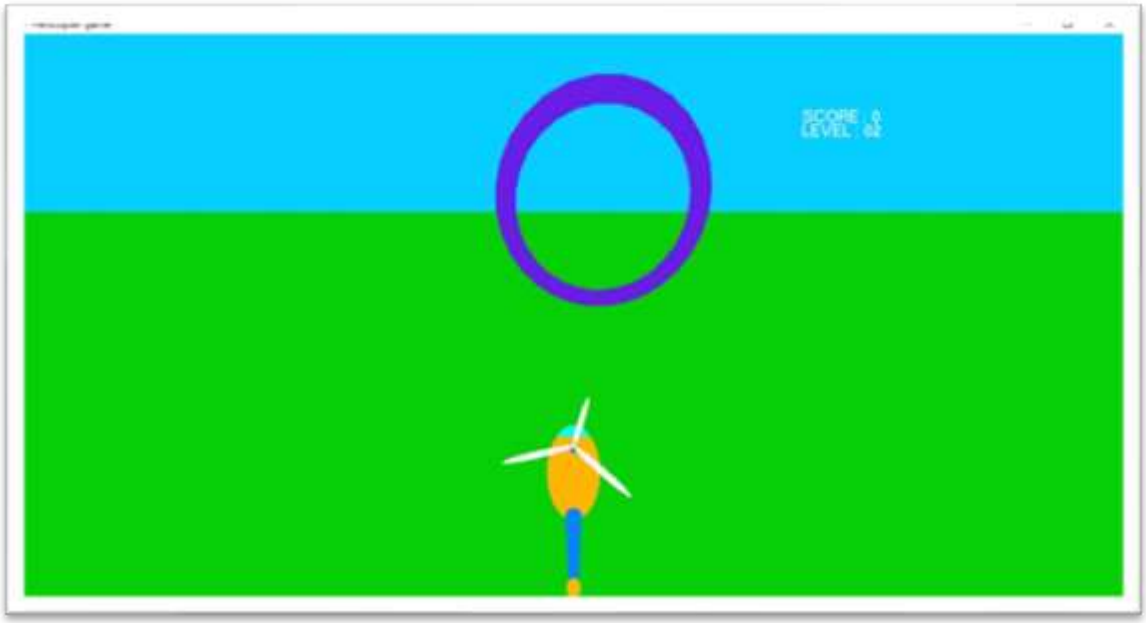


Fig 3.1.4: Downside navigated.

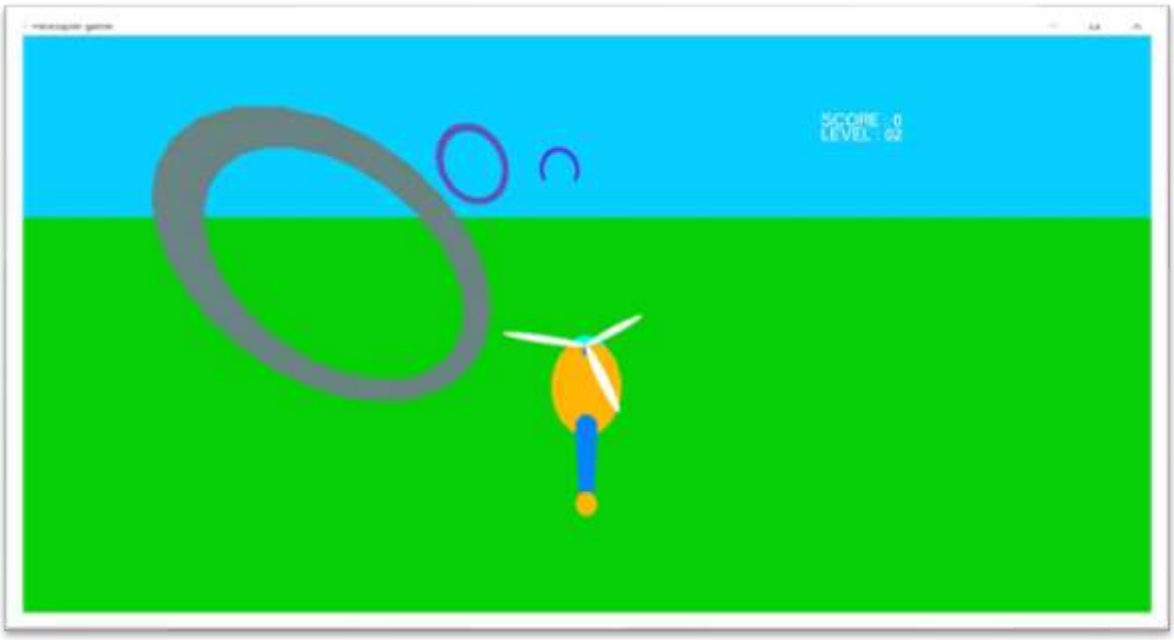


Fig 3.1.5: Right side navigated.

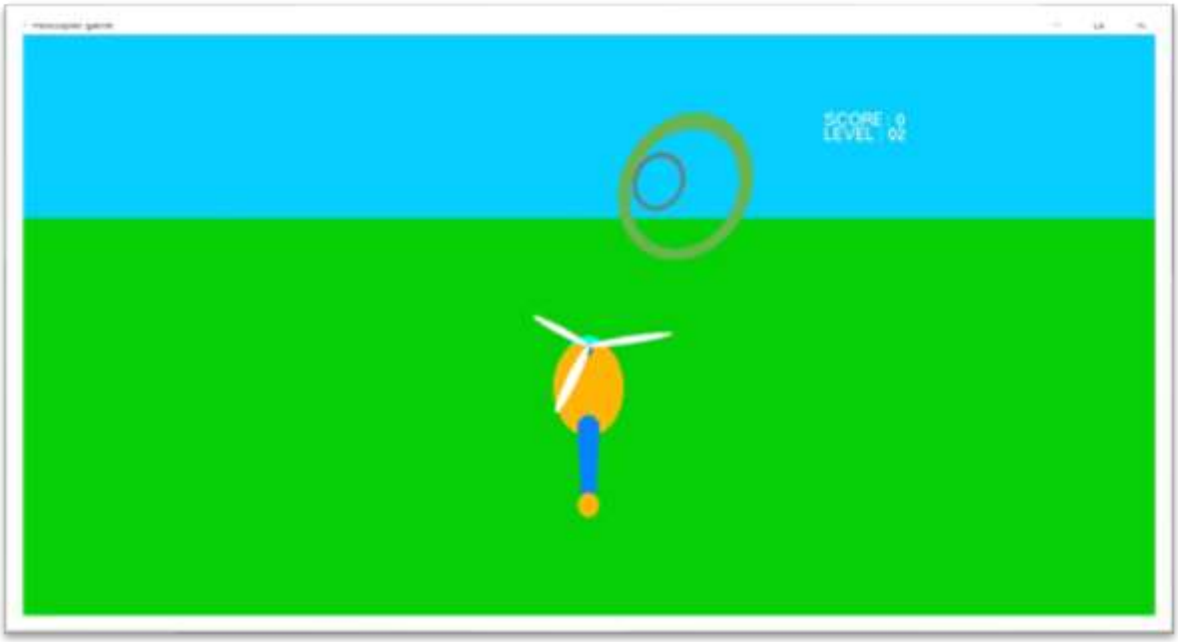


Fig 3.1.6: Left side navigated.

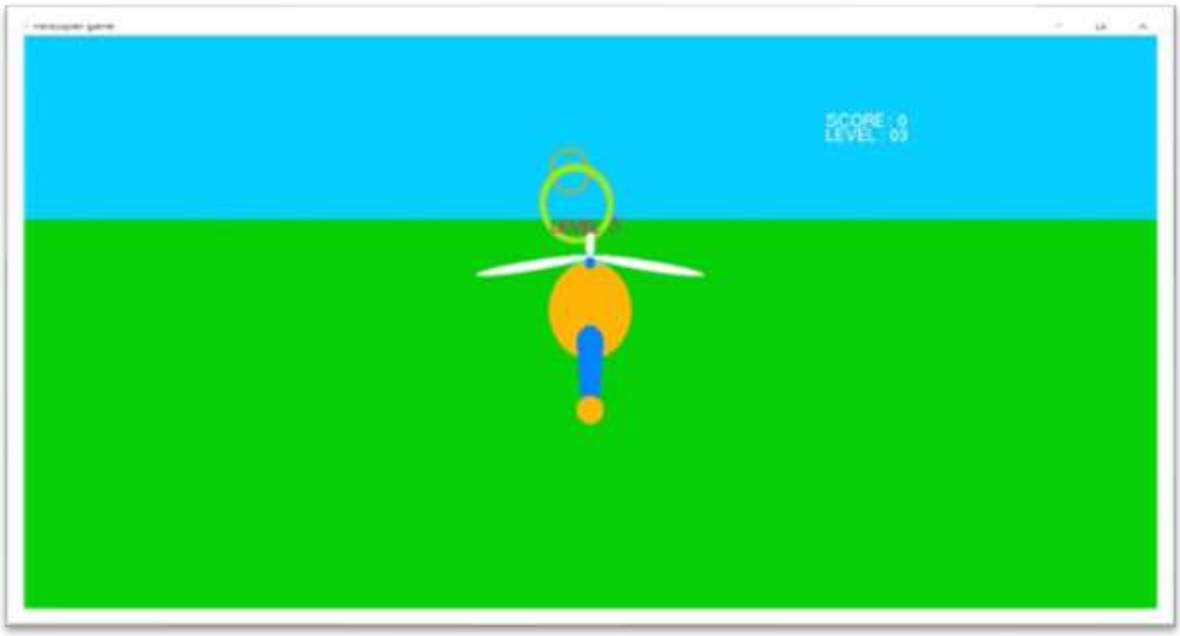


Fig 3.1.7: Level Increment.

3.2 Discussion

The project has been efficiently built. After all testing process, the program is now ready to be used. We have used primitive shapes available in the open GL library to create the Helicopter, Rings and the surrounding. With the help of keys in the keyboard the Helicopter can be navigated to any desirable direction. The keys the helicopter is directed through the rings, on every successful passing the score is incremented and as and when score increases the level and the speed of the game also increases.

CHAPTER – 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion

We developed a suitable graphics package to implement the skills learnt in theory classes and the exercises indicated in practical classes using OpenGL. We were able to develop an adorable, enjoyable and sophisticated **"3D HELICOPTER GAME"** using OpenGL.

4.2 Future Enhancement

- Different models of polygons can be Implemented and add more vehicles to the option and letting the user select one based upon his choice
- High end graphics can be used for better user experience.
- The complexity of the game can be increased by adding more obstacle on the way.

CHAPTER – 5

REFERENCES

- The Red Book-OpenGL programming Guide,6th edition
- Edward Angel Interactive Computer Graphics A Top-Down Approach with OpenGL, 5th edition, Addison and Wesley
- Donald Hearn and Pauline Baker: Computer Graphics- OpenGl version, 3rdedition, Pearson Education
- F.S. Hill Jr.: Computer Graphics Using OpenGL,3rdedition, PHI.
- Websites:

  http://www.opengl.org/registry/

  www.na.fs.fed.us

  www.openglforum.com

  http://www.esf.eduss

  www.vtucs.com